New functions for dynamic attributes in the multidatabase system MRDSM

Ph. Vigier, W. Litwin

INRIA

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 724

NEW FUNCTIONS FOR DYNAMIC ATTRIBUTES IN THE MULTIDATABASE SYSTEM MRDSM

Philippe VIGIER
Witold LITWIN

AOUT 1987

Nouvelles fonctions pour les Attributs Dynamiques dans le Système Multibase MRDSM

Ph. Vigier, W. Litwin

I.N.R.I.A 78153 Le Chesnay, France

RESUME

Un attribut dynamique est un attribut temporaire défini dans la requête par une transformation d'attributs réels. Dans le système multibase MRDSM, un attribut dynamique était connu seulement de la requête où il a été défini. Ensuite, il nécessitait la définition d'une application de l'attribut réel sur celui dynamique pour les recherches et de l'application inverse pour les mises à jour de l'attribut dynamique. Nous présentons deux extensions nouvelles de MRDSM. La première permet de définir un attribut dynamique pour plusieurs requêtes. La deuxième évite la définition de l'application pour les mises à jour si celle pour les recherches est définie par une formule calculable. Le système inverse alors automatiquement cette dernière définition. On ne connaît pas de SGBD offrant une possibilité similaire. Les principes proposés permettent en particulier de mettre à jour des vues comportant des transformations d'attributs. De telles mises à jour étaient jusque là en général considérées impossibles.

New Functions For Dynamic Attributes

In The Multidatabase System MRDSM

Ph. Vigier, W. Litwin

I.N.R.I.A., 78153 Le Chesnay, France

ABSTRACT

A dynamic attribute is a temporary attribute defined in a query as a transform of actual attributes. In the multidatabase system MRDSM, a dynamic attribute is known only to the query that defined it. Furthermore, it requires the definition of the mapping from the actual attributes to the dynamic one for retrievals and of the inverse mapping for the dynamic attribute updates. We describe two new capabilities of MRDSM related to the usage of dynamic attributes. The first, allows one to define a dynamic attribute for several queries. The second one, is a feature whereby one can avoid defining the inverse mapping if the mapping for retrievals is a computable formula. No other implementation of similar capabilities are known. In particular, the principles of mapping inversion also allow to update views with attribute value transformation. Such updates were up to now generally considered impossible.

1 INTRODUCTION

Databases (DBs) managed by a multidatabase system /LIT82/,/LIT84/, /LIT84a/ are generally autonomous. Similar data may differ with respect to value types or meanings. Thus, for example, prices may be in dollars ($) in one DB and in French francs (FF) in another. The differences create problems in expressing the queries. An equi-join on different currencies is for instance meaningless. A multidatabase manipulation language, should provide functions, making queries easily formulable in such an environment as well.

In /LIT86/, we introduced for this purpose the concept of a Dynamic Attribute. A dynamic attribute (DA) is a temporary attribute that is defined as a transform of some actual attribute(s) while formulating a query. It is unknown to any schema, unlike a virtual (a view) attribute. In a query, one may refer to the DA as if it was an actual attribute (with eventual restrictions on updates). The definition of a DA includes usually a mapping called retrieval mapping that defines the value of the DA from the actual one(s). Another mapping, called update mapping defines the actual values from the dynamic ones, if a dynamic value should be updated.

The concept was implemented within the relational multidatabase system MRDSM /LIT85/, /WON84/, /WON84a/. It basically allows to instantly convert actual value types into user defined value types and to refer to these new value types as if they were actual attributes. For instance, an actual price in FF may be instantly converted to a price in $ through a rate known to the query author only and used as if it was an actual price. In particular, the conversion may make data comparable and the joins formulable. One may also use it for instant homogenization of the result of a retrieval from different DBs. For instance, actual prices in FF in one DB and in $ in another DB may be presented all in $ or all in English pounds, etc.

Furthermore, one may easily guess "what if". For instance, one may dynamically experiment various exchange rates between $ and FF. One may also instantly define an attribute from several attributes (e.g. **salary = days * day_sal**). Finally, one may update

the dynamic value without having to express the corresponding actual update. One may thus directly increase the price in $, without formulating the actual update in FF.

In MRDSM, a DA exists only for the query that defined it. The experience shows however that several queries may need the same DA. Currently, the DA has then to be redefined in each query. Furthermore, it appeared that the system should be able frequently to determine automatically the update mapping. For instance if the retrieval mapping corresponds to a linear formula, like **price_$ = price_FF/6**, then the update mapping is **price_FF = price_$ * 6** and this formula may be found easily using the symbolic manipulation methods.

Below we discuss two extensions to MRDSM. The first one called hold option keeps a DA available for any number of queries in a session. The second one performs the inversion of retrieval mappings that are computable formulae or tables. This extension required the symbolic manipulation capabilities that are beyond the scope of current database systems and may require the software larger than many database systems themselves. We present two aspects of the corresponding implementation in MRDSM. On the one hand, MRDSM calls for service the existing symbolic manipulation system Macsyma /MAC83/ (400 K lines of PL1). In addition, it uses its own numerical methods that were implemented easily.

The principles of mapping inversion presented below apply also to updates of views with attribute value transformation. Up to now, it was generally believed that updates operations cannot be supported on such views (/DEL82/, p. 182, /DAT86/, 187),... This was one of the fundamental limitations on usage of views.

Section 2 reviews the concept of DA and its implementation in MRDSM. Section 3 discusses the hold option. Section 4 deals with the mapping inversion. Section 5 concludes the discussion.

2 DYNAMIC ATTRIBUTES

Let **D** be a dynamic attribute of a relation **R**. Let **N** be a set of actual attributes of **R** that we call *source attribute(s)* (for **D**). The values of **D** basically result from the retrieval mapping, called **M** ; **M : N -> D, n /-> d** ; where **n** = $(a_0, ..., a_m)$ are values in a tuple and

d is the value of **D** for this tuple. In MRDSM, **M** may be defined by a formula, a program or a table. The **D** values are updatable, if there is the mapping **M' ; M' : D -> N ;** called update mapping. This mapping has to be user defined in MRDSM for **M** defined by formulae, programs and usually for tables.

Dynamic attributes are one of the functions provided by the multidatabase manipulation language of MRDSM, called MDSL. For a detailed description of MDSL and its dynamic attributes see /LIT86-86a/. The general form of MDSL query with a DA is as follows.

-bd (<alias> <database name>)...
-range_s <semantic_variables>
-range (<tuple_variable> <relation designator>)...

-attr_d <name> **:** <value type>
-define by <mapping type>(<source>) = <mapping definition>
-updating <target> **by** <mapping type>(<source>) = <mapping definition>

-select <result>
-where <predicate>
retrieve / delete / modify <value_list> **/ insert** <tuples>

The clause **-bd** is optional. It defines the scope of the query and eventually short aliases for database names. The scope is the largest set of DBs that the query may address (the default is the currently open database). The clause **-range** defines tuple variables over relations. A relation designator is the relation name eventually qualified with the database name. It may be, as usually, a unique identifier. However, it may also be a multiple identifier or a semantic variable defined in **range_s** clause. In the latter case, the tuple variable is ranging over sets of relations. These relations may share the name (the multiple identifier) or may have different names declared then as the domain of a semantic variable. Such a query basically produces a set of relations that may be union non-compatible. See /LIT86/ for deeper discussion of these features of MDSL.

The clause **-attr_d** declares the name of a DA, with its value type. The type is either 'R' (numerical) or 'C' (characters). The clause **-define by** defines the retrieval mapping. The mapping type is either 'T' for a table or 'F' for an arithmetical formula, or P' for a program. The program may use any of the Multics system languages and should be precompiled. The mapping definition is a formula or a table or a program name.

The clause **-updating** defines the update mapping for DA updates. The target designates the actual attribute to be updated. The source contains for a table mapping the names of all other actual attributes the mapping concerns, if any.

The meaning of the clauses **-select** and **-where** as well as of the commands is the usual. One may use the DA in any clause and may declare any number of DAs per query.

Ex. 2.1. Let **michelin** be a database modeling the famous French restaurant guide Michelin. Let the restaurants be described as follows :

(1) **r (nr*,name,street,type,stars,avprice,tel),**

where '*' indicates the key attributes and **stars** is the restaurant rating ; **stars** = ' ', '*', '**', '***'. Let **tourist** be a database with a similar relation, except that prices are in $ and keys are independent from these in **michelin**. Assume that no two restaurants has the same name and street name. Consider the following queries of an American tourist who has exchanged his dollars at the rate of 6.2 F/$ he is the only to know :

(Q1) : Select the name, the address and the average price in $ of restaurants whose average price is the same according to both **michelin** and **tourist**.

The formulation of Q1 in MDSL is as follows :

(2.1)
```
-bd (b michelin) (c tourist)
   -range (t b.r) (s c.r)
-attr_d avprice : R
   -define by F(t.avprice) = avprice / 6.2
-select s.name s.street s.avprice
   -where (t.name = s.name) & (t.street = s.street)
 & (s.avprice = t.avprice)
```

The name **avprice** refers here to the actual attribute in the **define by** clause and to the dynamic attribute everywhere else.

(Q2) : Select from **michelin** the rating of restaurants whose price is under 10 $, under the same rate.

(2.2) **-bd (b michelin)**
-range (t b.r)
-attr_d avprice:R
-define by F(t.avprice) = avprice / 6.2
-select t.stars
-where (avprice < 10)

As far as we know, MDSL is the only database language that incorporates dynamic attributes. Other languages, like SQL in particular, allow the expression of queries **(Q1)** and **(Q2)** by using a value expression or by creating a view or a temporary relation. The corresponding form is however always more procedural and increases the possibility of an error. One has to repeat the value expression any time its result is required in the query formulation and a view or a temporary relation has to be created and dropped. Furthermore, none of languages allows to update the dynamic value.

The dynamic attribute name may in particular be a multiple identifier or a semantic variable designating a dynamic attribute in one database and an actual attribute in another database. The query that refers to it in the **select** clause is then evaluated as two queries : one providing the value of the dynamic attribute and the other providing the values of the actual one. This capability allows to easily homogenize heterogeneous values. See details in /LIT86/.

3 THE HOLD OPTION

3.1 The concept

The query **Q2** could be the follow-up query of **Q1** in the session. Many other queries could further follow **Q2**, using **avprice** as defined by **Q1**. Currently, MDSL requires a query using a DA to contain its definition. Thus, **avprice** would need to be redefined as many times as there are queries referring to. It is therefore useful to allow one to define a DA only once for all queries that need it in a session.

The **hold** option is a new option in MDSL intended for these needs. The DA is then called a hold attribute. If **D** is a DA, one declares **D** using the keyword **hold** in **-attr_d** clause :

-attr_d hold D : <value type>

The definition of **D** remains valid until the end of the session or until the command :

dadh D

(droop_ad_hold) is issued. Afterwards, the name **D** may be reused for another hold attribute. In addition, the command :

ladh

(list_ad_hold) lists all the current hold attributes.

3.2 Name conflicts

A hold AD may share the name of an actual attribute. Furthermore, the user may redefine the meaning of **D** for a query, for instance to change the exchange rate. To solve the corresponding name conflicts, the meaning of **D** is defined by the first condition to hold in the following order : (i) **D** created in the query (ii) the hold **D**, (iii) the actual **D**. The user may inverse (ii) and (iii) for a query, using the following clause prior to the **-select** clause :

-actual D.

Ex. 3.1 : If the query **(Q1)** had the **hold option**, then **(Q2)** could be simply :

(3.1)
```
-bd (b michelin)
    -range (t b.r)
    -select t.name t.street avprice
    -where (avprice < 10)
```

If the next query was for instance :

(3.2)
```
-bd (b michelin)
    -range (t b.r)
        -actual avprice
    -select t.name t.street avprice
    -where (avprice < 70),
```

it would mean that the user deals with FF. Then, the formulation :

(3.3) **-bd (b michelin)**
-range (t b.r)
-attr_d avprice:R
-define by F(t.avprice) = avprice / 6
-select t.stars
-where (avprice < 10)

would mean that the prices refer to the new exchange rate. In contrast, if next query was **(3.1)**, then it would mean that the hold rate : 1 $ = 6.2 FF) applies again. Finally, one should issue the statement :

dadh avprice

prior to any new **clause :**

-attr_d hold avprice:R

in the session. However, all hold meanings disappear with the end of the session and are unknown to any concurrent session.

3.4 Implementation overview

Hold attributes are stored in a temporary relation called Hold Option Dictionary. The dictionary and the query decomposition are managed in MRDSM by the Hold Option Processing Subsystem (HOPS). On one hand, HOPS transforms clauses with the **hold** option into these without the option and updates the dictionary if the attribute name is not in it. On the other hand, it adds the DA definition clauses, if the attribute name is already in the dictionary and there is neither the corresponding **-actual** clause nor the full definition of the DA in the query. Finally, it suppresses from the query the clauses **-actual** if any, and takes care of the commands **dadh** and **ladh**. Once HOAPS finishes the query processing, the query is in the form known to the DA processing subsystem that is then called. The dictionary is dropped when the session finishes.

4 UPDATE MAPPING COMPUTATION

4.1 Introduction

MRDSM requires the user to provide the update mapping, noted below **M'**, except for table mappings. **M'** is an inversion of the retrieval mapping, **M**. One may try to make this inversion automatic. When it works, the user may define only **M**. Two problems appear then :

- inversion computing.

It is simple when **M** is a table. It is more difficult for linear arithmetical formulae and even more for the polynomial formulae.It is currently usually impossible for programs.

- the choice of one inversion among several, when **M** is many-to-one mapping.

Let **M** be for instance a 2nd order polynomial, mapping values of an actual attribute **A** on **D**. Then, for any value of **A**, there are two potential values of **D**. The **M** definition alone, does not suffice for the choice. On the other hand, if **M** is an m-ary mapping and m > 2, then the formula defining **M** may not suffice for the choice of the attribute(s) to be affected by the update. Consider for instance the case of DA **salary** defined as **salary = days * day_sal**. One cannot know from the formula whether the change to **salary** should correspond to the change of **days**, or of **day_sal** or of both attributes.

There is no general solution to the above problems. The clause **-updating** stays therefore in MDSL for the cases when the system is unable to inverse **M** or the user prefers his own choice. In particular, one should use it for program mappings. The remainder shows methods avoiding the clause for table mappings and usual types of symbolic formulae.

4.2 Table mappings

Let $T(D, S) = T(d_1, s_1), T(d_2, s_2), \ldots T(d_k, s_k)$, be the table defining the retrieval mapping, where **S** denotes the actual attribute(s) and

D the corresponding DA. The mapping **M' : D -> S, M' : d /-> s** is defined as corresponding for each **d** to the first $T(d_i, s_i)$ such that $d = d_i$. Thus, it does not need to be bijective. If there is no such element of **T**, then the update is rejected.

Ex. 4.1 Consider the following DA, where the user converts the actual star rating to his own scale of rating (TOP, GOOD) :

-attr_d quality : C
-define by D(stars) = (TOP, *), (TOP,**), (GOOD,*)**

The mapping **M** is not bijective as both '**' and '***' ratings lead to 'TOP' rating. As defined, it means that the user wishes his update of **quality** to **TOP** to set **stars** to '***'.

4.2 Formulae

4.2.1 The problem

The general form of the retrieval mapping defined by a formula **F** is :

(4.1) $D = F(S_1, S_2, .., S_k)$ $\quad k > 0.$

The update mapping computation problem is as follows :

- let **d** be the retrieved dynamic value and s the corresponding actual source values :

(4.2) $d = F(s_1, s_2, .., s_k)$.

Let **d'** be the update value for **d** ; **d' <> d**. For each retrieved **d** and the corresponding **d'**, find the values **s'** for which :

(4.3) $d' = F(s'_1, s'_2, .., s'_k)$.

These values are then to be entered into the corresponding tuples.

For the computations of the retrieval mapping MRDSM uses the function CALC of Multics system. This function accepts parenthesed arithmetical formulae with the operators : + - * / ** . Below, we mainly consider only such formulae unless extensions to this assumption are explicitly stated. We further consider that basically one updates only S_1 which by default is the first attribute enumerated in the parenthesis in **define by** clause. The formula **(4.3)** becomes thus :

(4.4) $d' = F(s'_1, s_2, .., s_k)$

The choice which attribute should be S_1 is user subjective. In our example of **salary**, if the update to **salary** usually corresponds to a different number of working days, then the **define by** clause should be :

-define by F(days, day_sal) = day * day_sal.

MRDSM may nevertheless be asked to systematically display S_1 for confirmation or the choice of another actual attribute.

The formula **(4.4)** defines for each **d'** the equation :

(4.5) $d' = F(S_1, s_2, .., s_k)$

with the same single variable S_1. Each value s'_1 searched for is the root of **(4.5)**. Two methods appear for the solution of **(4.5)** :

(i) - the symbolic manipulation methods, finding the exact solution for each **d'**.

(ii) - the numeric methods, computing for each **d'** the roots of the expression :

(4.6) $0 = F(s'_1, s_2, .., s_k) - d'$

As it will appear, some cases require the combination of both approaches.

4.2.2 Symbolic manipulation

Even only for the considered CALC function class of formulae, the approach (i) requires rather complex expert system capability. The desired symbolic manipulation system should first be able to invert the linear formulae like :

avprice$ = avpriceFF / 6

into :

avpriceFF = $avprice$ * 6

It then should know to distribute the multiplications over additions at all levels of the formula. For instance, if the exchange rate is the official one plus 10 % and thus **(4.1)** is written as :

avprice$ = (avpriceFF / 6) + ((avpriceFF / 6) * (10 / 100))

then it should be factorized to the form :

avprice$ = avpriceFF * 1.1 / 6,

whose inversion is :

avpriceFF = avprice$ * 6 / 1.1.

Finally, since exponents are allowed, the system should have capabilities to solve **n**-th power equation. This is generally a complex task for **n > 2,** even only for polynomial equations.

To implement the corresponding symbolic manipulation system seems a huge task. The well known Macsyma system has for instance about 400 K instructions of LISP /MAC83/. Instead of writing during many years a symbolic manipulation subsystem for a multidatabase system, it seems more practical to use services of an existing one (note that

MRDSM is about 40 times smaller than Macsyma). Following this idea, a connection from MRDSM to Macsyma was implemented.

Through this connection, MRDSM passes to Macsyma the formulae **(4.5)** and calls the **solve** function of Macsyma. This function solves in general algebraic equations and returns the list of roots. If **k = 1**, then **solve** is called once per **d'**. Otherwise, it is used for each different value of s_j ; $1 < j =< k$; which is up to once per every tuple to update.

The interface was rather simple to implement, requiring the programming effort of about a month. This effort is without comparison to of implementing a symbolic manipulation system with the required capabilities. The solution allowed in addition to extend the class of the considered mappings to formulae with built-in Macsyma functions like the trigonometric or the logarithmic ones etc, unknown to CALC function. The principal limitation is that Macsyma is not able to solve all considered equations if **n > 4**, as it uses the algebraic approach. If the power of the equation is greater than quartic then the **solve** function tries to factor the equation into quadratic, cubic, or quartic formulae. If it does not succeed, only a partial solution is returned or no solution at all.

4.2.4 Numerical methods

Numerical methods solve systems of non linear equations, especially factorized to the polynomial expression of form **E(x) = 0**. They thus apply to **(4.5)**, provided its transformation to **(4.6)** and the factorization. MRDSM uses Bairstow method /DUR60/ that seems the best for the case. The method computes all the real and/or imaginary roots of a polynomial. Its principle is as follows :

- Let **P(X)** be the n-th power polynomial over variable X. The method seeks for the quadratic equation B_0 factorizing **P** into the expression:

$$P(X) = B_0(X) * Q_0(X).$$

One may then solve $B_0(X) = 0$ using quadratic formulae. The roots of B_0 are among the roots of **P** and Q_0 is a polynomial of power **(n-2)**. To find further roots of **P(X)**, one reapplies the factorization to Q_0 :

$$Q_0(X) = B_1(X) * Q_1(X).$$

The process continues until $Q_i(X)$ is a quadratic or a simple equation, depending on whether **n** is even or odd.

The main problem is to find B_i that divides **P** or Q_{i-1} exactly. To reach this goal, each B_i has the following form:

$$B_i(X) = X ** 2 - SM * X + PD$$

where **SM** and **PD** are respectively the sum and the product of two roots of **P(X)** (Q_{i-1}). The computation of **SM** and **PD** is done by the Raphson-Newton method with two variables /DUR60/.

4.2.5 Combination of the methods

The numerical method is able to solve equation that Macsyma could not solve if they are in a polynomial form. Macsyma transforms formulae into a polynomial form even if it cannot solve them itself. MRDSM combines therefore both methods in the following way :

- Macsyma is called for (**4.5**).
- It returns the roots or the polynomial it could not solve that is a factorized form of (**4.6**).
- If so, the Bairstow method is applied with the polynomial as **P(X)**.
- Macsyma is called again for the computation of the roots of each B_i.

4.3 Choice of the root

If **M'** is not a bijection, then the above procedure returns several roots, given each by a different formula. For instance, n-th power polynomial has n real and/or imaginary roots (Alembert's theorem). One has to determine which root should be chosen as the new actual value s'_1. Again, the formulation of **M** mapping alone does not allow to decide. The complementary general rule for the root choice in MRDSM is as follows :

The root to choose is :

(i) - either this given by the formula solving (4.2), i.e which applied to the retrieved dynamic value **d**, sets as the root the retrieved actual value s_1.
(ii) - or the i-th root provided by the Bairstow method, if s_1 is i-th root while solving (4.2) using this method.

The rule (ii) comes from the property of Bairstow method to determine the roots always in the same order for a given polynomial. The idea in the whole rule is that the existing formula or root number express hidden functional dependencies that an update operation in a database system should not affect.

Ex. 4.2. Consider :

-attr_d Q : R
-define by F(X) = X * X

where **X = 3** before the update and **Q = 16** after the update. The roots are :

$$X_1 = -SQRT(Q)$$
$$X_2 = +SQRT(Q).$$

For **X = 3, Q = 9**. For this **Q**, $X_1 = -3$ and $X_2 = +3$. Thus **X** should be updated to **X = 4**.

4.4 Implementation overview

4.4.1 Usage of Macsyma

MRDSM communicates with Macsyma through the absentee mode of Multics /MUL81/. This mode creates a process that executes commands from an input segment and puts the results in an output segment. MRDSM creates the input segment and then waits until the output file is created. More precisely, (i) the last command of the created process renames the output file into a particular name, and (ii) MRDSM looks for this file using the **exist file** command of Multics. The created process calls Macsyma, provides it with the corresponding input formulae and records the Macsyma outcomes in the output file. MRDSM reads then the output file and continues its session.

This mode of communication was chosen as the simplest one to implement. It was also of interest as the main mode for the cooperation of independently designed programs on Multics. It requires however the created process to wait until previous absentee processes end up. The waiting time acceptable for a prototype turned out to be sometimes rather too long for practical applications. The equivalent solution may nevertheless be a good approach for other systems, provided the possibility to create the process with high priority.

Another solution could be a LISP environment for MRDSM, written itself in PL1, that would allow to call Macsyma's functions directly or through Macsyma LISP environment. This approach could be faster on Multics, but presently Multics lacks of the capability for calls from a compiled program to a LISP interpreter. However, Macsyma is also slow by itself, especially when concurrently used. Detailed study of best performance in the discussed context remains to be done and was not the purpose of this work.

4.4.2 Bairstow method

The implementation led to the problem of limited precision of the computer, while the basic Bairstow algorithm considers the infinite precision. The propagation of the error due to the difference, amplified through the successive division of $\mathbf{Q_{i-1}(X)}$ by $\mathbf{B_i(X)}$, makes the polynomials $\mathbf{Q_i(X)}$ less and less correct. The errors in roots are more and more considerable, especially for high power polynomials.

To solve this problem MRDSM uses the optimal stopping test presented in detail in /VIG80/. The test provides the best **SM** and **PD** for the computer. The roots of the quadratic equation $\mathbf{B_i(X)}$ are tested as true roots of the polynomial $\mathbf{Q_{i-1}(X)}$ and if, needed, they are improved using the Newton method to obtain the best possible roots for the precision of the computer. See /VIG80/ for the discussion of the Newton method.

5 CONCLUSION

The hold option and the update mapping deduction are new functions that simplify the usage of dynamic attributes. We have described the corresponding concepts and the

technical solutions in the prototype system MRDSM. We showed that the update mapping deduction required symbolic manipulation methods. The corresponding implementation focused on the communication with an existing system that is Macsyma, instead of rebuilding a dedicated subsystem under MRDSM. This approach seems the most promising for future database systems. However, if a more restricted class of functions is considered, like only the most frequent linear formulae, then the corresponding system should be simple enough to be built in the database system.

This study seems to be the first ever done with respect to automatic deduction of update mapping. The presented principles apply not only to dynamic attributes, but also to updates of views with value transformation. Up to now, it was generally believed that updates operations cannot be supported on such views. This was one of the fundamental limitations on usage of views, although this concept is now known for two decades. Our results opens the way to database system free of this limitation.

The aim in the study was a theoretical solution to the problem and the proof of its technical feasibility. Many issues with respect to details and performance optimization are open. The class of the considered formulae should be enlarged, especially for new applications of databases to engineering or scientific applications. The proposed methods for the choice of the solution when the update mapping is not bijective may be refined, in particular through deeper run-time analysis of the actual data. One may also obtain better performance for more restricted classes of functions through usage of either other symbolic manipulation and numerical methods, or of tools like TK! Solver etc. on microcomputers /TK!84/. This is particularly important for practical applications, as a connection to Macsyma is generally not available.

A particular extension of deductive capabilities should concern the automatic unit conversion. The user would often need then to specify only the source and the target units. Both the retrieval and the update mappings would be determined by the system. This would further simplify the usage of dynamic attributes and again, of views as well, and should reveal highly appreciated.

Acknowledgments

We thank P. Scheuermann for helpful comments.

REFERENCES

/ABD83/ Abdellatif A., Multiple queries processing by a relational multidatabases system MRDSM, (in French), Res. rep., Tunis Univ., ed. INRIA, June 1983.

/ABD84/ Abdellatif A., Outer-join and standard function in the multidatabases system MRDSM, (in French), D.E.A. rep., Paris 6 Univ., ed. INRIA, June 1984.

/DAT86/ Date, C. An Introduction to Database Systems. Vol 1, 4th Ed. Addison-Wesley, 1986, 639.

/DEL82/ Delobel C., Adiba M. Relational Databases and Systems, (in French), Dunod, 1982, 449.

/DUR60/ Durand E. , numerical solution to algebraic equation (part 1), (in French), Masson & Cie, 1960.

/LIT82/ Litwin and all. SIRIUS, Systems for distributed data management, Distributed data bases, North-Holland, 1982.

/LIT84/ Litwin W., MALPHA : a relational multidatabase manipulation language, IEEE-COMDEC, Los Angeles, May 1984.

/LIT84a/ Litwin W., Concepts for multidatabase manipulation languages, JCIT-4, Jerusalem, June 1984.

/LIT85/ Litwin W., An overview of the multidatabase system MRDSM, ACM-85, Denver, Oct. 1985

/LIT86/ Litwin W. Vigier Ph., Dynamic attributes in the multidatabase system MRDSM, 2nd IEEE Conf. on Data Engineering, Los Angeles, Feb. 1986

/LIT86/ Litwin, W. Abdellatif, A. Multidatabase Interoperability. IEEE Computer, (Dec. 1986), 10-18.

/MAC83/ Macsyma reference manual, The Mathlab Group, Laboratory for Computer Science, MIT, Jan. 1983

/MUL81/ Multics Programmer's Manual, Commands and active functions, CII-Honeywell Bull, Ref. 68A2AG92 REV4, Nov. 1981.

/TK!84/ Tk!Solver User Manual. Software Art, 1984, 176.

/VIG80/ VIGNES J. , numerical algorithms analysis and implementation (part 2), (in French), Editions Technip, 1980.

/VIG84/ Vigier Ph., Processing dynamic attributes in the multidatabases system MRDSM, (in French), D.E.A. rep., Paris 6 Univ., Ed. INRIA, june 1984

/WON84/ WONG K. K., MRDSM : a relational multidatabase management system, 3rd international Seminar on Distributed Data Sharing Systems, North-Holland, 1984, 77-85

/WON84a/ Wong K. K., The design and implementation of a relational multidatabases management system MRDSM, (in French), Dr. Thesis, Univ. P. Sabatier (Toulouse), Mai 1984

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique